Kodi: Concrete Architecture Analysis

Link to Video

ROLES

Daniel Madan - Team Leader/Presenter/Reflexion Analysis/Use Cases

Daniel Solomon - Presenter/Abstract/Introduction/Lessons Learned/Conclusion

Ethan Fighel - Presenter/Reflexion Analysis/Use Cases

Zachary Kizell - Concrete Architecture

Brian Cabingan - Analyzed Subsystem

Brett Morris - Understand/Editor

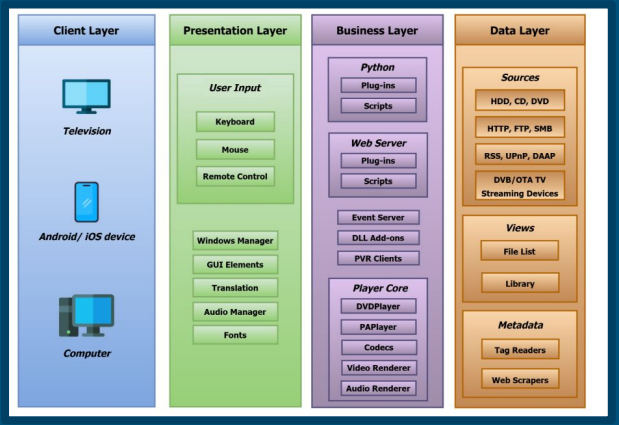
Table of Contents

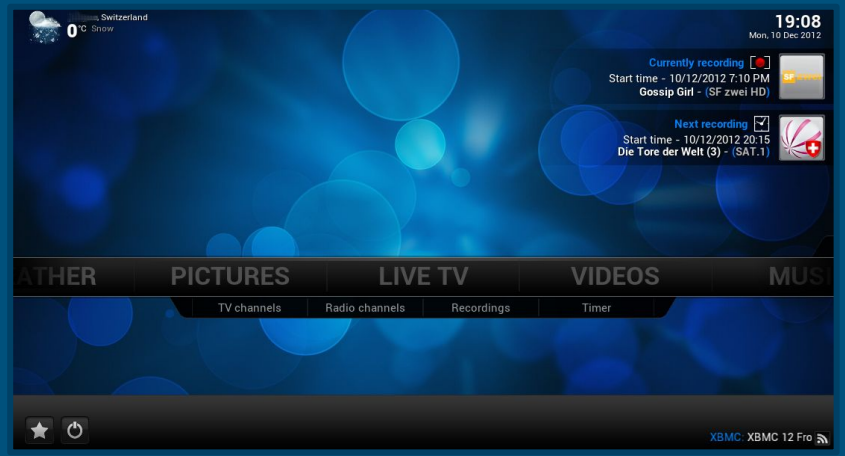
Abstract

I. Detailed study of Kodi's Video Player OSD subsystem's layered architecture and open-source deviations.
II. Use of Understand for source code analysis and subsystem component examination.
III. Contrast of theoretical architecture with Kodi's practical implementation.
IV. Exploration of use cases and architectural insights with key takeaways.

Introduction

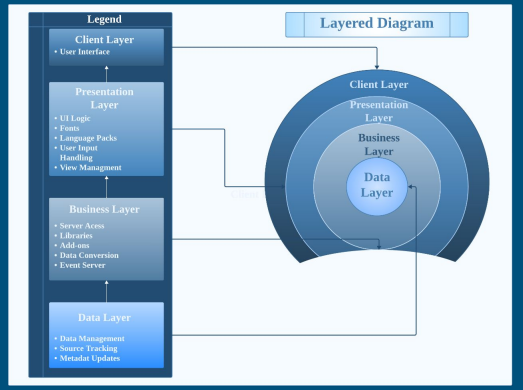
I. Kodi's OSD subsystem uses a four-layer architecture, influenced by its open-source nature.
II. Analyzed using GitHub, Understand, and Kodi Wiki, revealing component interconnectivity.
III. Key areas: User Input, GUI, and Player Core subsystems.
IV. Reflexion analysis showed deviations from the expected architecture, such as direct database queries and add-on-based subtitles.

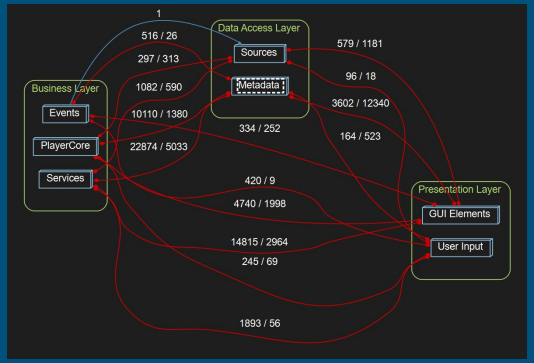




Concrete Architecture - Derivation Process

I. Analyzed Kodi's architecture using GitHub, Understand for dependencies, and the Kodi Wiki.
II. Dependency visualization helped clarify component relationships.
III. Folder/file examination and source code review informed architecture organization.
IV. The Kodi Wiki aided in defining subsystems within layers.
V. Kodi's four-layer architecture shows extensive inter-layer interactions, typical of open-source projects.





Subsystem Architecture of Video Player OSD

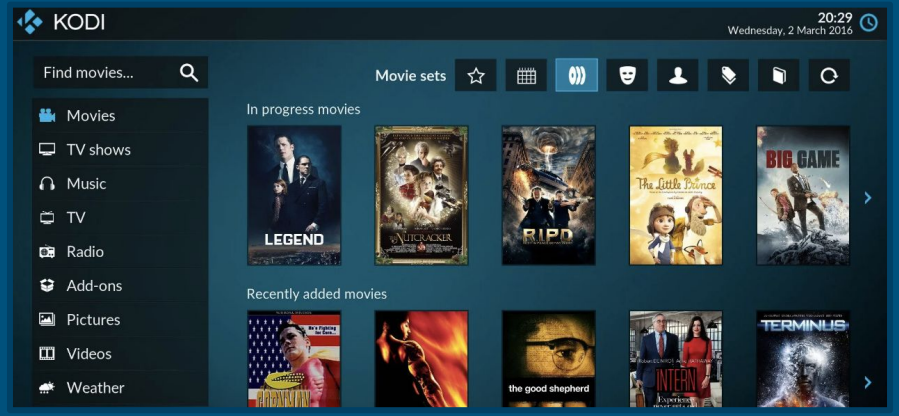
Architecture Overview - Presentation Layer

User Input Subsystem:

I. Captures and processes input for GUI, menus, and games.
II. Supports various hardware like keyboards and game controllers.
III. Uses an event server for device input handling.
IV. Features an English-to-Korean input converter.

GUI Subsystem:

I. Manages appearance of GUI elements: color, font, size.
II. Handles window management (fullscreen, windowed modes).
III. Uses distinct files for different GUI elements (e.g., radio buttons, text).
IV. Direct interaction with dbwrappers, games, events, and addons.



Architecture Overview - Business Layer

Player Core Subsystem:

I. Renders video/audio from sources like DVDs, digital files.
II. Supports retro game playback.
III. Uses PAPlayer for audio; integrates with GUI for display.
IV. Links with input, interfaces, and web server subsystems.

Python Subsystem:

I. Manages user-created Kodi add-ons.
II. Coded in C++ and Python.
III. Handles add-on installers, decoders, scrapers.

Web Server Subsystem:

I. Facilitates remote content management.
II. Allows multi-user access on a local network.

Architecture Overview - Data Layer

Sources Subsystem (Data Access Layer):

I. Manages media sources including local (HDDs, CDs, DVDs) and network (HTTP, FTP, SMB).
II. Handles files for network access, settings, and media source management.
III. Enhances network accessibility for media content.

Metadata Subsystem:

I. Stores and manages information for media and players.
II. Central to Kodi's architecture; includes key configuration files (e.g., ServiceBroker.cpp, Util.cpp).
III. Handles importing, exporting, storing, organizing, and transforming metadata.
IV. Critical for Kodi's efficient functionality.

Analyzed Subsystems

Application:

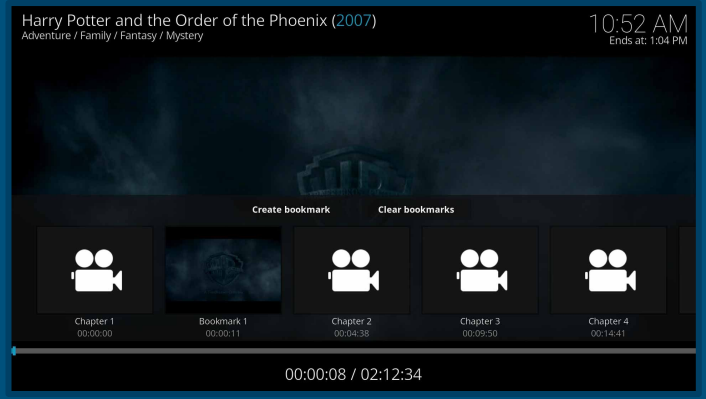
- Managed by Application.cpp and Application.h.
- Core functions: resolution matching, window updates, HDR toggle, video info display.

Bookmarking:

- Controlled by Bookmark.cpp and Bookmark.h.
- Features: adding bookmarks with episode/season details, time, and part number.

Dialogues:

- Involves various files (GUIDialogSubtitles.cpp, etc.) for different functions.
- Handles language retrieval, bookmark retrieval, and video settings.



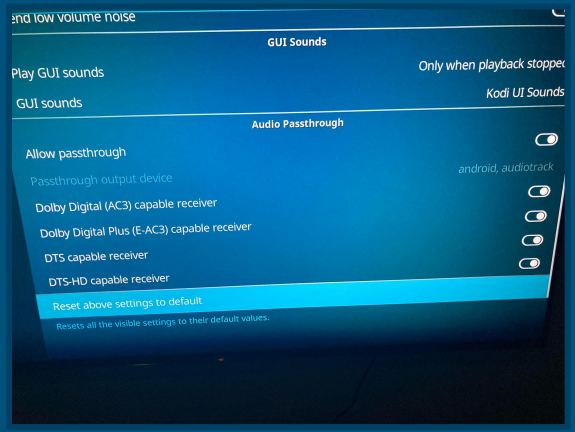
Analyzed Subsystems Cont’d

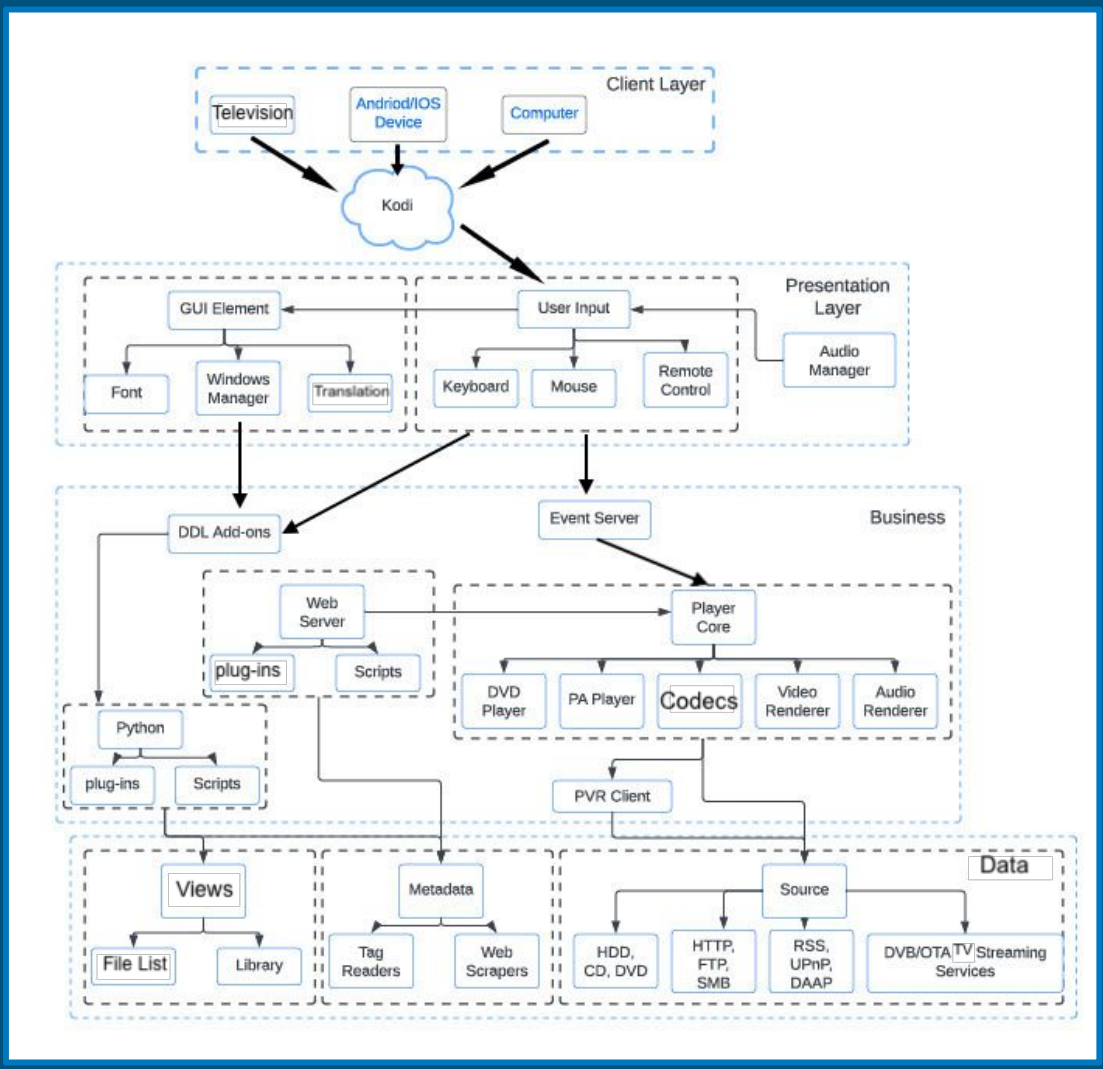
GUI Information:

- Managed by VideoGUIInfo.cpp, GUIMediaWindow.cpp, and PlayerGUIInfo.cpp.
- Displays video details (title, genre, etc.) and player information (volume, playback speed).

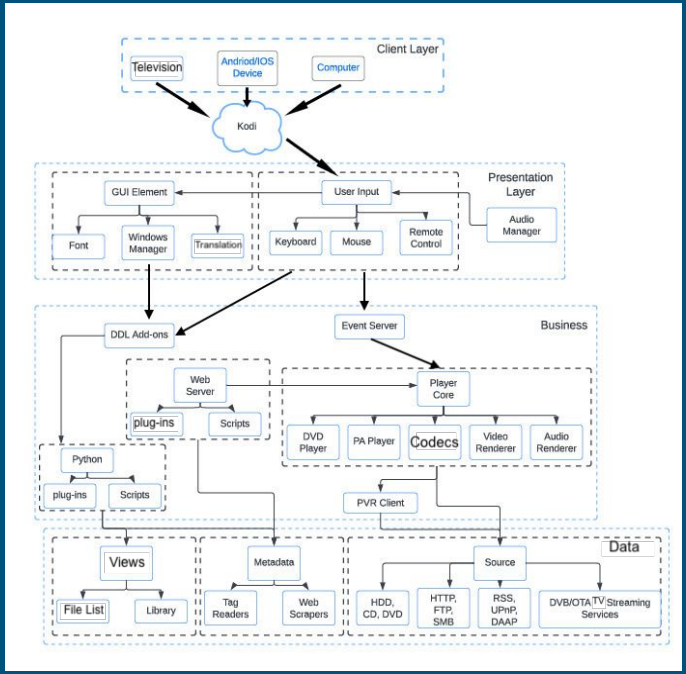
Other Components:

- FileItem.cpp and FileItem.h for file item types and resume points.
- ServiceBroker.cpp for function calls like GetGUI and GetMediaManager.




Client Layer
Television
Andriod/IOS Device
Computer
Kodi
Presentation Layer
GUI Element
Font
Windows Manager
Translation
User Input
Keyboard
Mouse
Remote Control
Audio Manager
Business
DDL Add-ons
Event Server
Web Server
plug-ins
Scripts
Player Core
DVD Player
PA Player
Codecs
Video Renderer
Audio Renderer
Python
plug-ins
Scripts
PVR Client
Data
Views
File List
Library
Metadata
Tag Readers
Web Scrapers
Source
HDD, CD, DVD
HTTP, FTP, SMB
RSS, UPnP, DAAP
DVB/OTA TV Streaming Services

Reflexion Analysis - High Level Architecture



Layered Architecture:

- Consistent interaction between business and presentation layers.

Layer Interactions:

- Presentation layer interacts directly with the data layer, bypassing the business layer.
- Absence of some view elements outside the conceptual architecture.

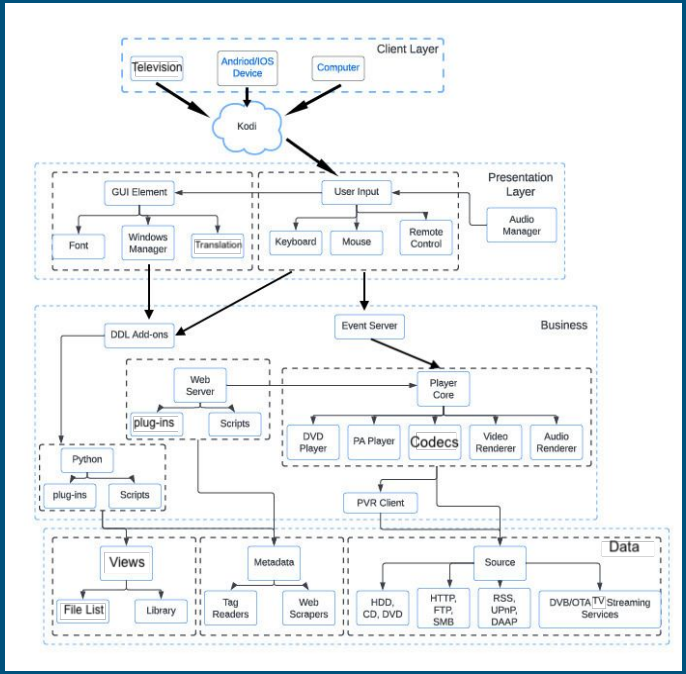
Open-Source Impact:

- Variations due to contributions from volunteer programmers.
- Often leads to deviations from intended architecture.

Source Code Analysis:

- Uncovered unexpected dependencies illustrating these variations.

Reflexion Analysis - Interaction Summary:



Presentation Layer Interactions:

Presentation Layer: GUI Elements ↔ Business Layer: Events:
Direct interaction to determine GUI element events and updates.

Presentation Layer: GUI Elements ↔ Business Layer:
PlayerCore: Interaction for editing toggleable GUI elements and layout adjustments based on media.

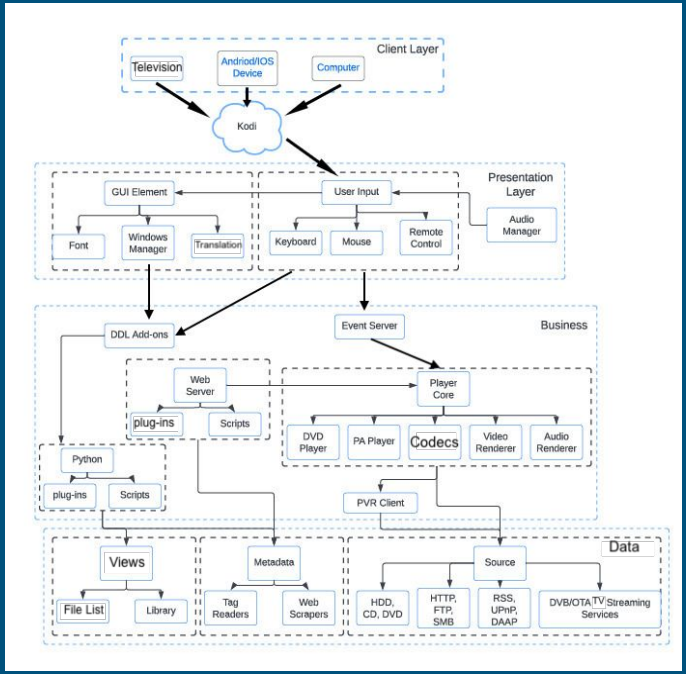
Presentation Layer: GUI Elements ↔ Data Access Layer: Metadata:
Sharing media information and updates to display to the user.

Presentation Layer: GUI Elements ↔ Data Access Layer: Sources:
GUI updates for file access methods selection.

Presentation Layer: User Input ↔ Data Access Layer:
Metadata: Configuration settings affecting menu interaction.

Presentation Layer: User Input ↔ Data Access Layer: Sources:
User input for adding and selecting media from the Sources database.

Reflexion Analysis - Interaction Summary Cont'd



Business Layer Interactions:

Business Layer: Events ↔ Data Access Layer: Metadata:
Event configuration based on media attributes.

Business Layer: Events → Data Access Layer: Sources:
Accessing media files when user selects them.

Presentation Layer: User Input → Business Layer: PlayerCore:
User interaction with media loading and display.

Business Layer: PlayerCore → Data Access Layer:
Metadata: Specific media aspects requiring configuration settings.

Business Layer: Events → Data Access Layer: Sources:
Playing media events using original files from the Sources database.

Video Player OSD Reflexion Analysis:

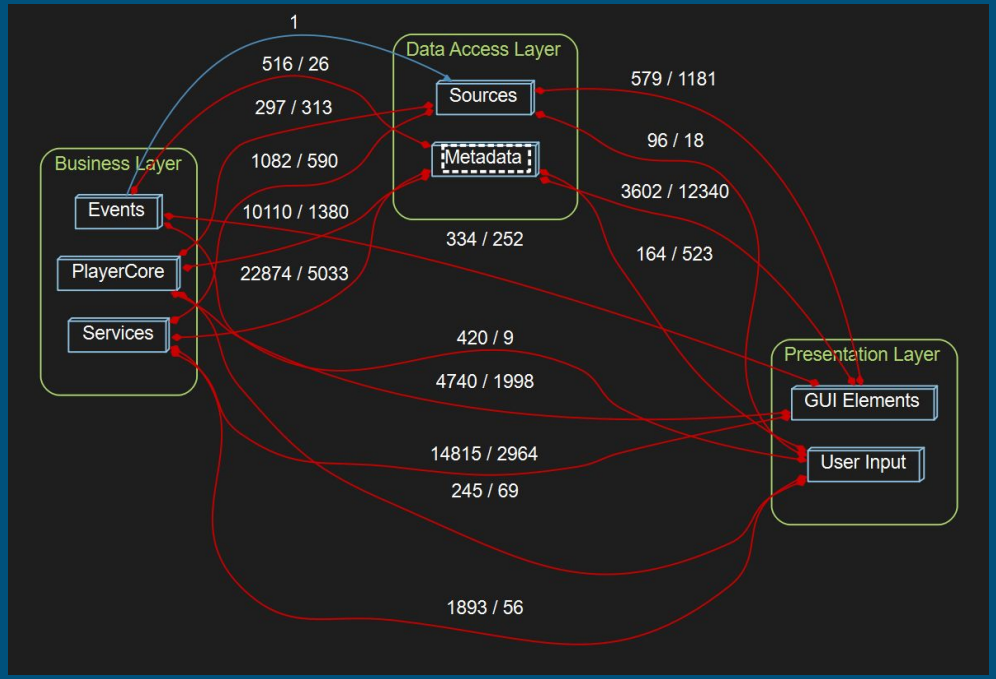


Figure 2. Subsystem Architecture of Video Player OSD

Video Player OSD Reflexion Analysis:

Reflexion Analysis on Video Player OSD:

- Identifies differences between conceptual and concrete architecture.

Video Information Retrieval:

- Direct database access for title, genre, duration.
- Bypasses expected web server mediation.
- Streamlines process, removing architectural intermediaries.

Subtitles Functionality:

- Utilizes add-ons for downloading from external sources.
- Allows direct loading of local subtitles.
- Avoids involving web server, enhancing efficiency.

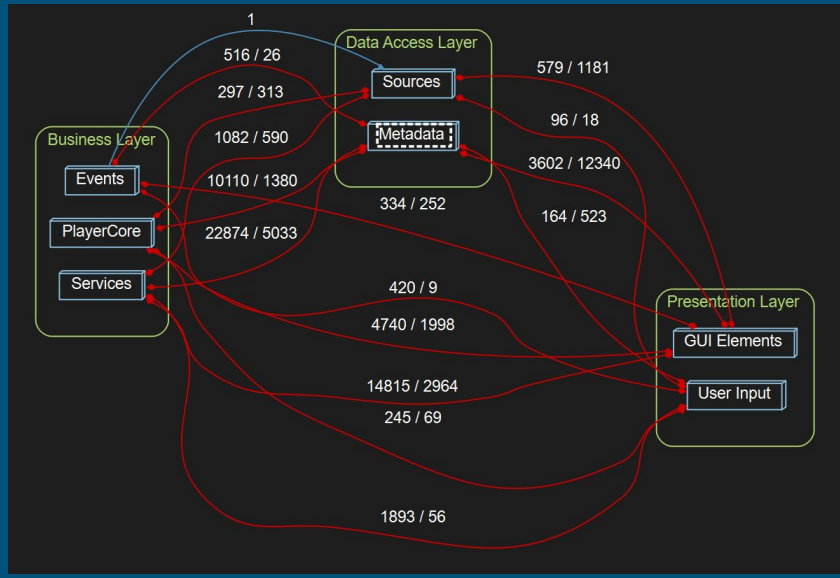


Figure 2. Subsystem Architecture of Video Player OSD

Use Cases

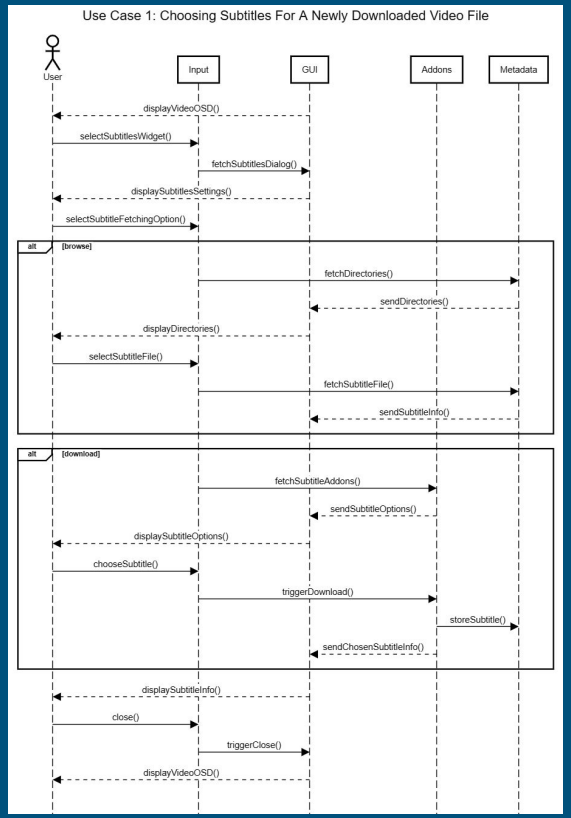


Figure 3: Use Case 1 Sequence Diagram

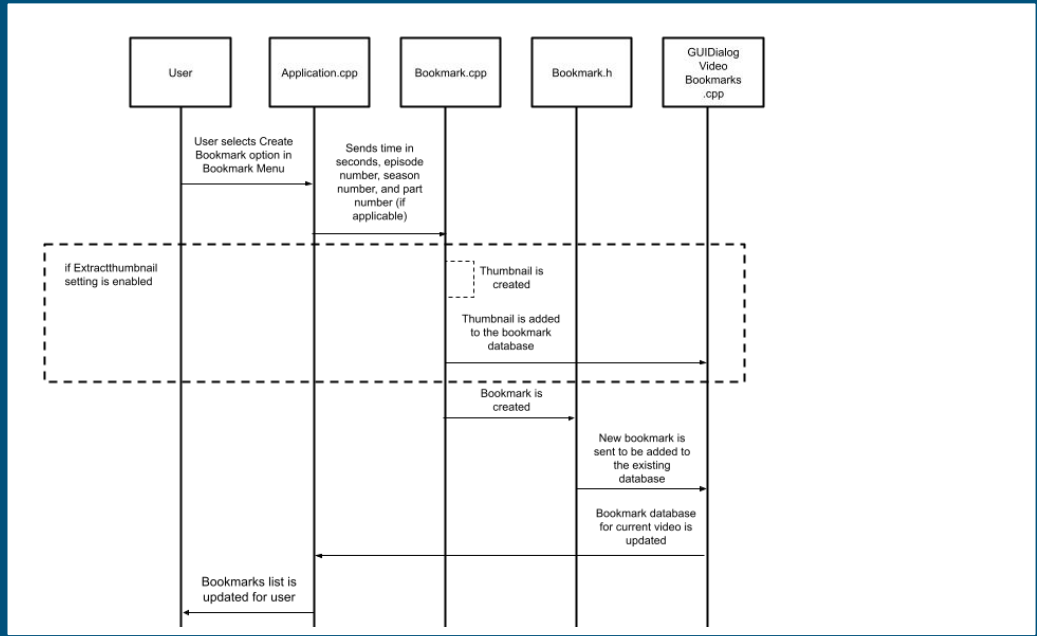


Figure 4: Sequence diagram for Use Case 2

Use Case 1

The user wants to choose subtitles for a newly download Video File.

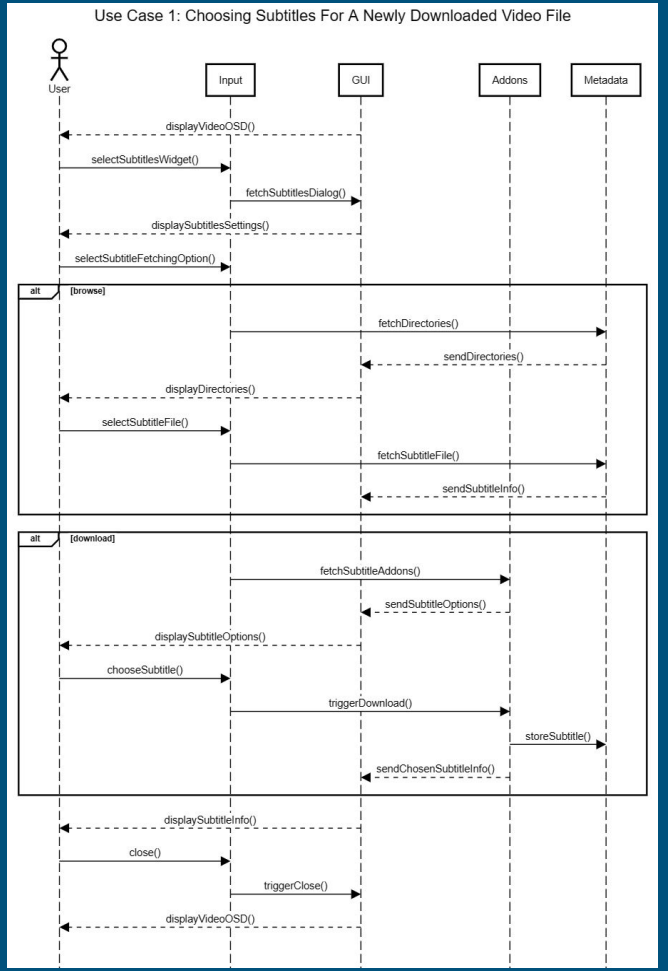


Figure 3: Use Case 1 Sequence Diagram

Use Case 2

A user is watching a movie on their computer and would like to create a bookmark so they can save the time code of the section they are on

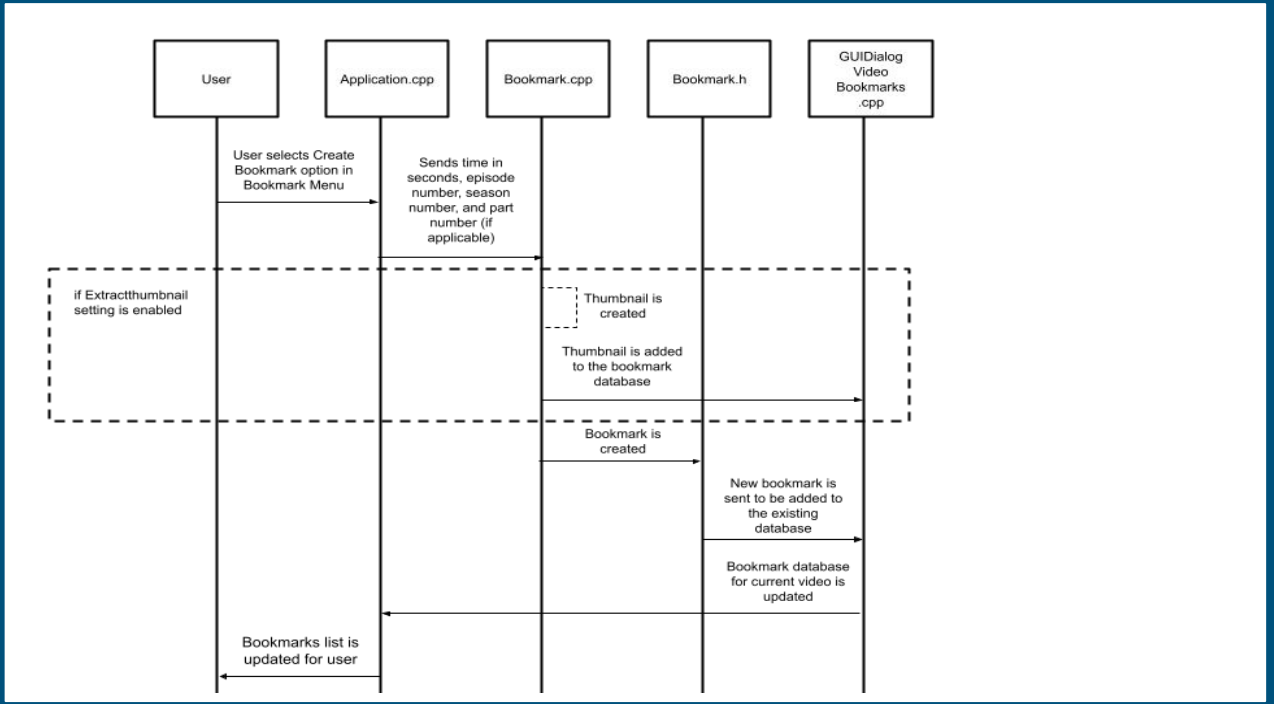


Figure 4: Sequence diagram for Use Case 2

Lessons Learned/Conclusion

- Kodi's architecture is non-traditional and layered, driven by its open-source nature.
- Reflexion analysis reveals architectural variances, including direct database access and unique subtitle management.
- Practical features like bookmarking and subtitle selection showcase the system's functionality.
- The report provides a detailed understanding of Kodi's architecture, emphasizing the influence of open-source principles on its design and highlighting the advantages and challenges of open-source software development within the context of Kodi's media center software.

Conclusion

- Detailed analysis of Kodi's Video Player OSD subsystem, focusing on its layered, open-source architecture.
- Utilized GitHub source code, Understand for dependency tracking, and the Kodi Wiki.
- Identified unique features and deviations from traditional models.
- Explored User Input, GUI, and Player Core subsystems.
- Highlighted architectural discrepancies through reflexion analysis.
- Discussed adaptability and practical design in open-source development.
- Presented as a resource on Kodi's architecture and open-source software design.

Thank You For Listening